Hector²

Variables formulées

DEFINITIONS

Qu'est-ce qu'une variable formulée

Une variable formulée est une variable dont les valeurs, pour chaque sujet, n'ont pas été saisies à la main dans Hector ou importées depuis un fichier texte (comme les variables d'origine), mais calculées selon une formule rédigée par l'utilisateur dans le langage des Formules, en s'appuyant sur des variables déjà existantes, qu'elles soient d'origine ou elles-mêmes formulées. Typiquement, et pour donner un exemple très basique, pour calculer d'un seul coup pour tous les sujets la somme de deux variables :

```
# score_total
: score_épreuve_A + score_épreuve_B ;
```

Les variables `score_épreuve_A` et `score_épreuve_B` préexistent ; la variable `score_total` est créée est calculée par l'exécution de la formule.

L'utilisateur d'un tableur pourra faire l'analogie avec les formules et les fonctions, sauf que dans Hector on se réfère à des variables désignées par un nom plutôt qu'à des colonnes de cases repérées par leur position dans la feuille de calcul. De plus, la syntaxe des formules dans Hector est de beaucoup plus claire que celles des tableurs, où l'amoncellement des parenthèses et des points-virgules rend souvent les choses peu lisibles.

Les variables formulées sont créées dans la Page FORMULES. Une fois créées, elles s'utilisent comme les variables d'origine dans tous les traitements. Dans les listes de variables, leur intitulé apparaît en *italique*, et non en caractères ordinaires. Les variables formulées peuvent être *rejouées*, par exemple si on a modifié des données dans les variables d'origine, ou si on a saisi de nouveaux sujets[1]. Pour ce faire, chaque variable formulée garde le souvenir de sa formule, et peut la réexécuter à la demande. Les formules peuvent aussi être sauvegardées dans des fichiers de type texte et d'extension .fch situés dans un répertoire contigu au fichier-corpus, qui porte le nom du corpus avec une extension .co, comme fichiers-compagnons. Cette disposition est particulièrement recommandée en cas de formules sur des collections, mais cela permet aussi de se constituer des bibliothèques de formules à réemployer ou à utiliser comme modèles.

Comme les variables peuvent être rejouées à la demande, mais ne le sont pas si on ne le demande pas, le lien entre variables d'origine et variables formulées peut être qualifié de semi-dynamique[2]. On dispose aussi d'une commande qui fait rejouer toute une série de variables sélectionnées.

---

[1] La notion de corpus *terminé* n'est pas pertinente avec Hector : on peut toujours ajouter des sujets et des variables.

[2] C'est une différence avec les anciennes variables dérivées, qui sont obsolètes à partir d'Hector², mais sont conservées dans la version professionnelle pour des raisons de compatibilité avec des corpus susceptibles d'être exploités plusieurs années de suite. Cette modification fondamentale, qui semble faire perdre à Hector un peu de sa rigueur et de sa sécurité, apporte en compensation une très grande souplesse dans la gestion des variables. Elle nécessite cependant de la part de l'usager une bonne compréhension de ce qu'il fait et une certaine rigueur dans la gestion de son stock de variables. Une autre différence est que la syntaxe du langage des formules est entièrement différente de celle du langage de dérivation : elle se veut beaucoup plus souple et naturelle d'emploi. Elle couvre toutes les fonctionnalités de la dérivation, et en ajoute quelques autres. De plus, la manière dont elle est programmée permet l'ajout relativement aisé de nouvelles fonctionnalités en cas de besoin, alors que c'était pratiquement impossible dans la dérivation.

LA PAGE FORMULES

La page FORMULES est organisée de la manière suivante :

- à droite les panneaux des variables[3] et des collections, comme un peu partout dans Hector
- à gauche, en haut l'éditeur de formules et en bas le panneau des commentaires d'Hector
- entre les deux, plutôt sur la droite, un panneau de commande surmontant l'arbre des mots-clefs.

Le panneau des variables (et celui des collections)

Cliquer sur un intitulé de variable dans ce panneau a pour effet d'insérer cet intitulé dans l'éditeur de formules, à l'emplacement du curseur. On remarque que si l'intitulé comprend des espaces, ceux-ci sont remplacés dans la formule par le caractère '_', parce que l'espace est dans le langage des formules le séparateur universel des mots : donc quelque chose qui comporte un espace à l'intérieur n'est pas un mot mais plusieurs.

Si on contrôle-clique le nom d'une variable nominale ou ordinale, ses étiquettes s'affichent dans une liste en haut du panneau de commande, et pourra y cliquer pour les insérer.

Si on clique sur le nom d'une collection, ce nom s'insère dans le texte de la formule, comme pour les noms de variables.

L'éditeur de formules

C'est un éditeur de texte ordinaire, c'est-à-dire qu'on y clique pour y placer le curseur d'écriture, on peut y sélectionner, copier, couper et coller[4]. C'est là que l'utilisateur écrit ou modifie ses formules.

Tout ce qui existe *déjà*, noms de variables comme noms de collections ou étiquettes de variables à texte, *ne doit pas* être tapé au clavier par l'usager, mais inséré par clic dans la liste correspondante. En effet, l'orthographe des mots doit être intégralement respectée : un « a » n'est pas un « à » ni un « A ».

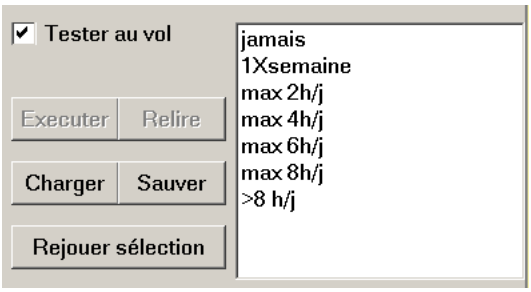
Le panneau des commentaires

Ce n'est pas l'utilisateur qui écrit là, mais Hector, qui réagit au fur et à mesure à ce que l'utilisateur écrit (à condition que la case [Tester au vol] soit cochée) en résumant ce qu'il a compris. On reviendra en détail là-dessus dans la rubrique Recherche des erreurs, mais on peut déjà noter que tant que le commentaire d'Hector est coupé par une ou plusieurs lignes vides, c'est que la formule est inachevée ou incorrecte.

[3] Eventuellement remplacé par le VariArbre, dispositif de structuration visuelle du corpus, dans la version professionnelle.

[4] Y compris en relation avec un fichier de texte extérieur, ouvert sous le Bloc-Notes ou sous un traitement de texte.

Le panneau de commande



Nous avons laissé de côté pour le moment le cadre [Corpus relationnel], en relation avec des fonctions très spécialisées, et qui fera l'objet d'une explication détaillée à la rubrique « Base de données relationnelles ».

La partie droite du panneau contient la liste des étiquettes de la dernière variable nominale ou ordinale qu'on ait contrôle-cliquée[5]. Ici il s'agissait d'une variable relative à l'usage d'internet.

La case à cocher [Tester au vol] indique si Hector teste la formule en cours d'élaboration *à chaque caractère tapé*. Si ce n'est pas le cas, aucune réaction d'Hector quand on tape la formule ; quand on a fini, on coche la case pour obtenir l'interprétation de ce qu'on a écrit.

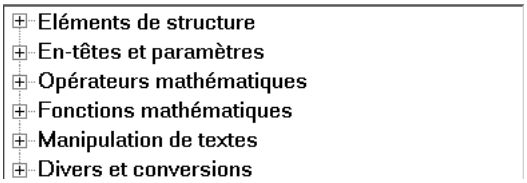
Le bouton [Exécuter] n'est actif que si une formule complète est prête à l'exécution. Dans ce cas, Le panneau des commentaires contient un seul bloc de texte qui commence par FORMULES …, ce qui signifie qu'Hector a pu interpréter le texte pour le réduire à un objet exécutable bien formé.

Le bouton [Relire] n'est actif que si une variable formulée (dont le nom est en italique) est sélectionnée dans la liste des variables. Si c'est le cas, cliquer le bouton [Relire] réinstalle dans l'éditeur de formules celle qui a servi à créer la variable, aux fins de modification ou simplement pour la rejouer. Droite-cliquer le nom de la variable formulée a le même effet.

Le bouton [Charger] permet de relire dans l'éditeur de formule le contenu d'un fichier texte à extension .fch, normalement[6] situé dans le répertoire-compagnon du corpus. Symétriquement, le bouton [Sauver] ouvre un dialogue de sauvegarde dans un fichier .fch du contenu actuel de l'éditeur de formules.

Le bouton [Rejouer sélection] a pour effet de rejouer toutes les variables formulées[7] comprises dans une sélection de variables[8]. Les variables sont rejouées dans l'ordre de la liste : il faut donc avoir fait attention[9] à éviter de se référer dans une variable formulée à une autre variable formulée placée plus bas dans la liste, car la seconde ne serait pas encore mise à jour au moment de la réexécution de la première ; ou alors il faut répéter la manœuvre une seconde fois.

L'arbre des mots-clefs



[5] C'est-à-dire cliquée, la touche Ctrl étant maintenue enfoncée.

[6] On peut naviguer autant qu'on veut et placer ses fichiers dans les endroits les plus abscons, mais il est souvent conseillé d'adopter en matière de gestion des fichiers une discipline d'acier.

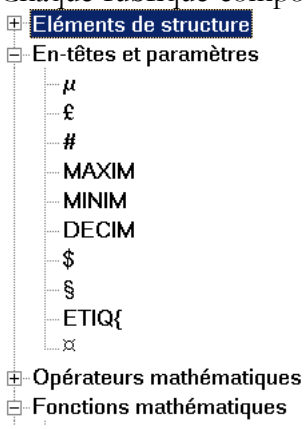
[7] Et celles-là seulement, les non-formulées sont ignorées.

[8] Obtenue dans la liste des variables en traînant la souris, en cliquant le début de la liste puis en shift-cliquant la fin, ou encore en contrôle-cliquant successivement et séparément plusieurs variables.

[9] Cela n'est pas formellement interdit, car dans certains cas limite c'est ce dont on a besoin, notamment si on veut simuler un calcul récursif.

On n'en montre ici que le début. Au départ ne sont affichées que les rubriques, véritables « familles » de mots-clefs, qui sont les mots réservés par Hector dans le langage des formules. Dire que ce sont de mots réservés implique qu'on n'a pas le droit de s'en servir pour désigner autre chose : de toutes façons Hector les interprétera toujours de manière prioritaire comme étant les siens.

Chaque rubrique comporte au départ un signe +, qui signifie qu'on peut déployer son contenu :



Cliquer sur l'un des mots-clefs provoque son insertion dans la formule en cours de rédaction, suivi d'un espace pour le séparer du mot suivant. Là aussi, il vaut mieux cliquer un nom de fonction que de se voir refuser l'exécution parce que le mot est mal écrit[10].

L'aide en ligne des formules

De plus, double-cliquer sur un mot-clef ou sur sa tête de rubrique provoque l'affichage de l'aide en ligne pour cette rubrique. Cette aide en ligne contient un rappel de ce que fait le mot-clef, rappel sommaire mais généralement suffisant pour lever une hésitation.

[10] Les mots-clefs sont interprétés par Hector indifféremment en majuscules et en minuscules, mais on suit dans ce manuel l'usage conseillé de les écrire en majuscules pour mieux les distinguer du reste du texte.

LE LANGAGE DES FORMULES

Mécanismes fondamentaux

Séparateurs

Dans le langage des formules, le séparateur universel est l'espace, auquel sont assimilés les tabulations, sauts de ligne et de paragraphe. Plusieurs séparateurs successifs sont traités comme un seul. L'usager est donc invité à user largement des séparateurs pour aérer son texte.

En conséquence, l'unité syntaxique est le mot, défini comme tout ce qui ne contient pas de séparateur.

Espaces dans un texte

Comme l'espace est un séparateur, un texte ne peut contenir d'espaces internes. Pourtant, des textes comme des noms de variables ou des étiquettes *doivent* pouvoir contenir des espaces. La solution est dans l'utilisation du caractère _ (également dit *souligné* ou *underscore*, sous la touche 8 en haut du clavier), qui remplace dans les formules les espaces internes aux mots, tout en étant traité ailleurs, par exemple dans un résultat, comme un espace. Conséquence : le caractère _ ne peut plus être distingué d'un espace. Prix peu élevé pour une simplification drastique de la syntaxe par économie de séparateurs de texte, tels que des accolades, des crochets ou des guillemets.

Entités élémentaires reconnues par l'éditeur de formules

Pour celui-ci, un mot est une des choses suivantes :

- C'est un mot qu'il connaît parce qu'il fait partie de son vocabulaire, autrement dit un *mot-clef*. Un mot-clef peut ne comporter qu'un seul caractère. Ces mots sont détaillés dans la liste en bas du panneau de commande, ils y sont en majuscule mais les versions en minuscules sont également valables. Si on clique un mot-clef dans sa liste, il se place à l'emplacement du curseur dans l'éditeur de formules, et ça évite les fautes d'orthographe, qu'Hector n'apprécie vraiment pas. L'usager ne peut pas utiliser les mots-clefs pour désigner autre chose que ce qu'ils ont : on les appelle donc aussi *mots réservés*.
- Si ce n'est pas un mot-clef, alors serait-ce un nombre explicite, une *constante numérique* ? Pour cela, il suffit qu'il s'agisse de quelque chose qui est écrit conformément à la syntaxe classique des nombres en algèbre et sur les calculatrices, en utilisant indifféremment comme séparateur décimal le point comme les anglo-saxons ou la virgule comme les français. 14 -12 0,56 .56 32e3 (32×10^3) sont des constantes numériques correctes, mais 1000,567.32 1000.567,32 n'en sont pas, car il ne peut y avoir de séparateur de milliers puisque le point et la virgule sont également admis comme séparateur décimal.
- Si ce n'est ni un mot-clef, ni une constante numérique, alors ça tombe dans la catégorie ultime : c'est une *constante textuelle*. « machin » est une constante textuelle, mais aussi « 45truc+ » si c'est tout collé.

Muni d'une liste d'entités élémentaires, la LISTE GENERALE, Hector va s'ingénier à les interpréter et à les combiner pour obtenir des valeurs et des formules bien bâties. Pour cela, il dispose d'un système de *règles d'interprétation*, lesquelles font l'objet de priorités : certaines s'appliquent *avant* les autres.

Les priorités vont de 0 à 9, des plus urgentes aux moins urgentes : aucune règle d'une certaine priorité n'est exécutée tant qu'il reste à exécuter des règles plus prioritaires. L'exécution des règles

va dans le sens de la réduction[11] de la liste générale, jusqu'à ce qu'elle ne contienne plus qu'un seul élément : une formule correcte et applicable.

Valeurs explicites et références

Dans Hector, les variables et les collections portent des noms, des intitulés. Ce sont des textes, mais il ne sont pas à prendre au pied de la lettre : si par exemple il existe une variable « Age » dans un corpus, la présence du mot Age dans une formule ne doit pas renvoyer au texte « Age », mais bien à une variable numérique qui peut prendre une valeur différente pour chaque sujet.

Dans le langage de formule, tout texte est examiné pour vérifier s'il ne s'agit pas d'une *référence*, c'est à dire du nom d'une variable ou d'une collection, plutôt que d'un simple texte à prendre tel quel.

Ainsi, si l'on tape dans l'éditeur :

```
Ag
```

étant entendu qu'il n'existe aucune variable ni collection de ce nom, et que ce n'est pas un mot-clef du langage, Hector commente ainsi, juste en-dessous (si on a coché la case [tester au vol]) :

```
TEXTE ... Ag
```

Que s'est-il passé ? L'interpréteur de formules a trouvé une CONSTANTE TEXTUELLE « Ag », ni mot-clef ni nombre ni rien d'autre, et, par défaut, a appliqué la règle :

```
3 TEXTE ← CONSTANTE_TEXTUELLE
```

La règle se lit comme ceci :

- Le petit chiffre au début est l'indice de priorité de la règle. Pour l'usager, généralement cela n'a aucune importance. Dans quelques cas cela en aura : ils seront signalés.
- A gauche de la flèche, le produit, ou résultat, de la règle, autrement dit le type de l'objet résultant de l'application de la règle.
- A droite de la flèche, ce qui a déclenché l'application de la règle

En d'autres termes, l'interpréteur « se dit » : si, appliquant les règles de priorité 3, je trouve une CONSTANTE_TEXTUELLE, je la requalifie en TEXTE.

A vrai dire, si une CONSTANTE_TEXTUELLE « traîne » encore au niveau 3, c'est qu'elle n'a pas fait l'objet d'une préemption[12] par une règle de niveau 0 à 2.

TEXTE est un des trois types principaux de valeur que manipule le langage de formules. Les deux autres sont NOMBRE et LOGIQUE (vrai, faux).

Un TEXTE, on le verra, peut provenir comme ici d'une CONSTANTE_TEXTUELLE, ou texte explicite, mais aussi d'une variable à valeurs du genre Texte (nominales, ordinales, textes libres), ou de toute opération plus complexe dont le résultat est du genre Texte.

Si, dans l'exemple ci-dessus, on prolonge encore d'une lettre la frappe :

```
Age
```

En supposant qu'il existe une variable numérique « Age » (mais pas « age » ni « âge »), le commentaire devient :

```
NOMBRE ... Age
```

parce que la CONSTANTE_TEXTUELLE « Age » a été identifiée comme le nom d'une variable numérique dans le corpus en cours, et que donc à l'exécution de la formule elle renverra, *pour*

---

[11] A noter que « réduction » est aussi la véritable signification du mot « algèbre ». Cf. Al Kwarismi, (780-850), *Kitâb al-jabr wa al-muqâbala.*

[12] Phénomène par lequel une règle de niveau élevé (proche de 0) s'empare d'un terme, le convertit à son gré, et le soustrait ainsi aux règles suivantes.

chaque sujet à son tour, une valeur numérique qui pourra entrer dans d'autres combinaisons de calculs.

Cette puissante règle de recherche systématique des références à des variables et à des collections facilite énormément l'écriture des formules. Toutefois, on pourrait souhaiter conserver à « Age » sa simple valeur de texte, même si existe une variable de ce nom. Le moyen de le faire existe (quelle règle n'a pas d'exceptions ?) : il suffit de placer un " (guillemet ou *double-quote*) devant, avec un espace entre deux :

```
" Age
```

est commenté par :

```
TEXTE ... Age
```

A noter que le " a disparu au passage. On peut ainsi « textifier impérativement » des nombres ou des mots-clefs. :

```
" 6 " +
```

est commenté :

```
TEXTE ... 6
TEXTE ... +
```

Il faut admettre que c'est assez troublant, et qu'il faut bien avoir compris ce qu'on fait quand on emploie cette exception, pourtant utile dans quelques cas spécifiques.

De la même manière, la règle

```
3 NOMBRE ← CONSTANTE_NUMERIQUE
```

fait que

```
65
```

est commenté

```
LISTE GENERALE ...
  NOMBRE ... 65
```

Un NOMBRE, peut provenir comme ici d'une CONSTANTE_NUMERIQUE, ou nombre explicite, mais aussi d'une variable à valeurs du genre Nombre (numériques, calendaires), ou de toute opération plus complexe dont le résultat est du genre Nombre.

Après assimilation de ces mécanismes de base, on peut passer à la présentation systématique, du simple au complexe, des règles du langage.

Arithmétique élémentaire

L'arithmétique du langage est très comparable aux règles usuelles :

```
5 NOMBRE ← NOMBRE + NOMBRE
```

Le caractère + est ici en gras pour rappeler que c'est un mot-clef. De même :

```
5 NOMBRE ← NOMBRE - NOMBRE
```

Les priorités d'opérateurs prennent tout leur intérêt avec l'apparition de la multiplication, notée par l'astérisque, et de la division, notée par la barre oblique :

```
4 NOMBRE ← NOMBRE * NOMBRE
4 NOMBRE ← NOMBRE / NOMBRE
```

Avec un niveau 4, multiplication et division s'effectuent *avant* l'addition et la soustraction : ainsi une expression telle que :

```
6 / 3 + 4 * 2
```

n'a aucune ambiguïté : elle équivaut à (6 / 3) + (4 *2) et à rien d'autre. Si l'on n'est pas trop sûr des priorités ou qu'on souhaite s'en affranchir, les parenthèses sont là pour ça :

```
1 NOMBRE ← ( NOMBRE )
```

et on peut donc passer outre les règles de priorité grâce à une structure comme 6 / (3 + 4) *2.

Cette fois, l'addition aura lieu *avant* la division et la multiplication, car elle est « protégée » par les parenthèses, qui se résolvent par une règle très prioritaire (1).

La division est l'occasion d'aborder la question des erreurs. On sait que la division par zéro n'a pas de sens. Que se passe-t-il si l'on écrit 6 / 0 ? Apparemment rien, mais le résultat de l'opération n'est pas un nombre, mais **RIEN**, une non-valeur, qui peut aussi être notée **@**, et qui correspond aux informations qu'on n'a pas (une case vide dans le tableau de saisie des données) ou qu'on en sait pas calculer (comme dans 6 / 0). Donc, dans le langage des formules, l'erreur est traitée en silence, sans protestations ni scandale ; parfois, si des variables formulées possèdent étrangement peu de sujets concernés (ayant une valeur), il est utile de soupçonner ce genre de phénomène.

3 NOMBRE ← **--** NOMBRE

Cette règle calcule l'opposé d'un nombre. Pour éviter la confusion avec la soustraction, le signe - est doublé.

2 NOMBRE ← NOMBRE ****** NOMBRE

L'exponentiation, marquée par une double astérisque, possède une priorité plus élevée encore. Si la puissance à laquelle il faut élever le premier nombre est 2 (carré), on préférera la règle suivante :

2 NOMBRE ← NOMBRE **²**

Attention, le ² qui note l'exposant[13] doit être détaché du nombre par un espace.

Contagion des non-valeurs

L'absurdité est contagieuse : la présence d'un terme non-défini (non-valeur, RIEN, @) parmi les opérandes arithmétiques force un résultat lui-même indéfini : 5 + on ne sait quoi = on ne sait quoi.

Fonctions sur les nombres

Assez classiquement, les fonctions sont traduites par un nom de fonction, suivis d'un couple de parenthèses qui encadrent le ou les arguments de la fonction. Attention ! Pour éviter la confusion avec une paire de parenthèses ordinaires, la première parenthèse est *collée* au nom de fonction :

1 NOMBRE ← **ABS(** NOMBRE **)**

1 NOMBRE ← **TRONQUE(** NOMBRE **)**

1 NOMBRE ← **ARR(** NOMBRE **)**

1 NOMBRE ← **ARR(** NOMBRE NOMBRE **)**

La première fournit la valeur absolue d'un nombre. La seconde le tronque s'il est réel, c'est à dire supprime ses décimales. La troisième l'arrondit à l'entier, selon la règle dite de l'arrondi commercial : à 4 on descend, à 5 on monte. La dernière est une variante où le second nombre précise le nombre de chiffres après la virgule qu'on désire conserver.

1 NOMBRE ← **LN(** NOMBRE **)**

1 NOMBRE ← **LOG10(** NOMBRE **)**

1 NOMBRE ← **LOG2(** NOMBRE **)**

1 NOMBRE ← **EXP(** NOMBRE **)**

Dans l'ordre, le logarithme naturel ou népérien, les logarithmes en base 1à et en base 2. Enfin l'exponentielle e^x. Dans tous les cas, des arguments inacceptables pour la fonction provoquent une erreur silencieuse, avec pour résultat une valeur indéfinie du genre **RIEN**.

[13] C'est une fantaisie rendue possible par l'existence de ce caractère à l'extrême gauche des claviers français.

Aléatoires et divers numériques

1 NOMBRE ← **UNIFORME(** NOMBRE NOMBRE **)**

1 NOMBRE ← **NORMALE(** NOMBRE NOMBRE **)**

1 NOMBRE ← **ALEA(** NOMBRE NOMBRE **)**

UNIFORME renvoie un aléatoire réel situé entre les deux nombres avec une répartition équiprobable sur l'intervalle (dernière valeur exclue) ; si les deux nombres ne sont pas en ordre croissant, ils sont échangés au préalable. NORMALE renvoie une valeur aléatoire réelle extraite d'une distribution normale dont les nombres donnés en arguments sont la moyenne et l'écart-type. ALEA renvoie un aléatoire *entier* compris entre les deux nombres préalablement arrondis si nécessaire, tous deux inclus.

1 NOMBRE ← **SUJET**

Le nombre renvoyé est le numéro implicite du sujet (à partir de 1).

Opérations logiques élémentaires

Les valeurs fondamentales du domaine logique peuvent être introduites par deux mots-clefs qui font office de constantes logiques.

3 LOGIQUE ← **VRAI**

3 LOGIQUE ← **FAUX**

Une valeur LOGIQUE peut aussi provenir d'une variable de ce type, mais aussi et le plus souvent de la confrontation d'une valeur d'un type quelconque à un *prédicat* approprié (voir ci-après).

1 LOGIQUE ← **(** LOGIQUE **)**

Les parenthèses, en logique comme ailleurs, permettent de contourner les règles de priorité ou de s'en assurer en cas de doute.

Opérateurs et priorité

3 LOGIQUE ← **NON** LOGIQUE

4 LOGIQUE ← LOGIQUE **ET** LOGIQUE

5 LOGIQUE ← LOGIQUE **OU** LOGIQUE

5 LOGIQUE ← LOGIQUE **OUX** LOGIQUE

La première règle fournit la négation d'une valeur logique, les autres les opérateurs classiques ET, OU et OUX (ou exclusif). On notera le parallélisme entre le système de priorité des opérations logiques et celui des opérations arithmétiques de base : le NON est prioritaire sur tout, le ET est prioritaire sur les deux OU et OUX. Cela implique par exemple que pour calculer la négation d'une formule complexe, il faut la mettre entre parenthèses : NON (a ET b OU c) équivaut à NON ((a ET b) OU c) à cause de la priorité du ET, mais NON a ET b OU C équivaut à ((NON a) ET b) OU c, c'est-à-dire quelque chose de très différent.

Logique ternaire

Du fait de l'existence, pas étonnante dans une base de données, de valeurs indéfinies (cases vides, etc.), la contagion des non-valeurs ne peut pas être appliquée directement aux opérations logiques. En effet dans a OU b, si a est VRAI, le résultat sera obligatoirement VRAI, que b soit VRAI, FAUX ou indéfini. En d'autres termes, VRAI OU on ne sait quoi est toujours VRAI. De même, FAUX ET on ne sait quoi est toujours FAUX. En revanche, VRAI ET on ne sait quoi est indéfini, ou, si l'on préfère, indécidable.

On est donc obligé d'admettre qu'on se trouve devant une logique *ternaire*, puisqu'elle possède trois valeurs de vérité distinctes : VRAI, FAUX et RIEN (incertain, on ne sait pas).

On dresse donc ci-dessous les tables de vérité des opérateurs logiques selon ces trois états :

Table de vérité de (NON a) selon les valeurs de a :

a	NON a
VRAI	FAUX
RIEN	RIEN
FAUX	VRAI

Table de vérité de (a ET b) selon les valeurs de a et de b :

	b = VRAI	b = RIEN	b = FAUX
a = VRAI	VRAI	RIEN	FAUX
a = RIEN	RIEN	RIEN	FAUX
a = FAUX	FAUX	FAUX	FAUX

Table de vérité de (a OU b) selon les valeurs de a et de b :

	b = VRAI	b = RIEN	b = FAUX
a = VRAI	VRAI	VRAI	VRAI
a = RIEN	VRAI	RIEN	RIEN
a = FAUX	VRAI	RIEN	FAUX

Table de vérité de (a OUX b) selon les valeurs de a et de b :

	b = VRAI	b = RIEN	b = FAUX
a = VRAI	FAUX	RIEN	VRAI
a = RIEN	RIEN	RIEN	RIEN
a = FAUX	VRAI	RIEN	FAUX

Quoique assez déroutante de prime abord, cette logique ternaire est, à y bien réfléchir, plus conforme au raisonnement quotidien que celle du tiers exclu.

Opérations sur les textes

Opérations de base

On ne peut évidemment pas parler d'arithmétique sur les textes, mais on a quand même quelque chose qui y ressemble un peu :

```
5 TEXTE ← TEXTE + TEXTE
```

Il s'agit de la *concaténation* : le texte résultat est constitué des deux textes arguments, mis bout à bout.

```
5 TEXTE ← TEXTE +_ TEXTE
```

Cette variante de la concaténation insère un espace entre les deux textes d'origine dans le texte résultant. Elle équivaut à + _ +.

A toutes fins utiles, les parenthèses existent aussi pour le TEXTE :

```
1 TEXTE ← ( TEXTE )
```

Fonctions sur les textes

Comme les autres fonctions du langage, elles sont caractérisées par un nom qui s'achève par la parenthèse ouvrante.

1 NOMBRE ← **TAILLE(** TEXTE **)**

Le résultat est un nombre qui indique combien de caractères comporte le texte (espaces[14] compris).

1 TEXTE ← **MAJUSCULES(** TEXTE **)**

1 TEXTE ← **MINUSCULES(** TEXTE **)**

Le texte est transcrit en majuscules ou en minuscules, respectivement.

1 TEXTE ← **COMPACTE(** TEXTE **)**

remplace dans un texte une série de caractères identiques par un caractère suivi de la longueur de la série si elle est supérieure à 1 (par exemple xxxxxx par x6, mais y pour y). Cette fonction renvoie une valeur indéterminée si le texte d'origine contient des chiffres.

1 TEXTE ← **DECOMPACTE(** TEXTE **)**

opération réciproque de la précédente (x6 devient xxxxxx). Ces deux fonctions trouvent toute leur utilité dans le maniement de séquences (version professionnelle/recherche).

1 LOGIQUE ← **PRESENCE(** TEXTE TEXTE **)**

1 NOMBRE ← **POSITION(** TEXTE TEXTE **)**

1 NOMBRE ← **OCCURRENCES(** TEXTE TEXTE **)**

PRESENCE renvoie VRAI si le premier texte est inclus dans le second, FAUX sinon. POSITION renvoie un entier qui est la position du premier texte dans le second s'il y est inclus (à partir de 1), 0 sinon. OCCURRENCES renvoie le nombre de fois distinctes ou le premier texte est compris dans le second : OCCURRENCES(aa aaa) vaut 1 et non pas 2, car après avoir reconnu les deux premiers a, l'algorithme continue au delà de la zone identifiée.

1 TEXTE ← **PREMIERS(** NOMBRE TEXTE **)**

1 TEXTE ← **DERNIERS(** NOMBRE TEXTE **)**

1 TEXTE ← **RESUME(** NOMBRE TEXTE **)**

PREMIERS renvoie un texte composé des *nombre* premiers caractères du texte argument. DERNIERS fait la même chose, mais en partant de la fin. RESUME renvoie un texte d'au plus NOMBRE caractères, en résumant le TEXTE selon un algorithme qui ne conserve que le début des mots, introduit par une majuscule interne (le résumé en 10 caractères de « la définition de la fonction » est « LaDéDeLaFo ») ; si, si, ça peut être utile.

1 TEXTE ← **EXTRAIT(** NOMBRE NOMBRE TEXTE **)**

1 TEXTE ← **REMPLACE(** TEXTE TEXTE TEXTE **)**

EXTRAIT fournit un sous-texte commençant dans le texte argument à la position indiquée par le premier *nombre*, sur une longueur indiquée par le second. Si la position est inférieure à 1 ou supérieure à la longueur de l'argument, la valeur renvoyée est un texte vide (de longueur nulle). Si l'extrait va théoriquement plus loin que l'extrémité de l'argument, il s'arrête à cette extrémité. REMPLACE remplace précisément le premier texte par le second dans le troisième. Si le premier est contenu dans le second, le résultat est indéfini, car il s'étendrait indéfiniment.

1 TEXTE ← **N°(** NOMBRE **)**

N°(renvoie l'étiquette définie explicitement par ETIQ{ (voir ci-après, en-têtes ordinales et nominales) et de rang NOMBRE dans la série, rang compté à partir de 0 et non de 1.

[14] Plus exactement pseudo-espaces codés « _ ».

Conversions

Le mécanisme de conversion prend pour argument une valeur d'un certain type, et fournit comme résultat une valeur d'un type différent :

$_1$ TEXTE ← **TEXTE(** LOGIQUE **)**

Une logique vraie est traduite par le texte « oui », une logique fausse par le texte « non ». Dans toutes les conversions, un argument indéterminé produit un résultat indéterminé.

$_1$ TEXTE ← **TEXTE(** NOMBRE **)**

Le nombre est converti en texte selon les conventions usuelles. 34,78 devient « 34,78 ».

$_1$ LOGIQUE ← **LOGIQUE(** TEXTE **)**

Tout dépend du premier caractère du texte argument. S'il vaut 'O', 'o', 'T', 't', 'V', 'v' ou '1', le résultat est VRAI. Si ce premier caractère vaut 'N','n','F','f' ou '0' (zéro), le résultat est FAUX. Si le premier caractère est quelque chose d'autre, le résultat est RIEN. Une utilisation courante de cette fonction est de récupérer des données issues de bases de données où les logiques sont transcrites par des mots en français, en anglais ou en allemand.

$_1$ LOGIQUE ← **LOGIQUE(** NOMBRE **)**

Le résultat est FAUX si le nombre est nul (égal à zéro), VRAI dans tous les autres cas.

$_1$ NOMBRE ← **NOMBRE(** TEXTE **)**

Le résultat est un nombre si le TEXTE était une représentation correcte de nombre, RIEN si la conversion est impossible.

$_1$ NOMBRE ← **NOMBRE(** LOGIQUE **)**

Le résultat est 1 pour VRAI, 0 pour faux.

Dates

$_1$ NOMBRE ← **DATE(** TEXTE **)**

Les dates sont en fait des nombres de jours écoulés depuis une date repère conventionnelle. La conversion réussit si le texte est une date possible, au format jj/mm/aaaa (ex : 18/06/1946), sinon le résultat est RIEN. Le format jj/mm/aa est aussi accepté, mais le millésime est incertain.

$_1$ NOMBRE ← **DI(** NOMBRE **)**

renvoie 0 (Lundi) à 6 (Dimanche) si l'argument est une date

$_1$ NOMBRE ← **JOUR(** NOMBRE **)**

renvoie le quantième du jour dans le mois si l'argument est une date

$_1$ NOMBRE ← **MOIS(** NOMBRE **)**

renvoie 0 (Janvier) à 11 (Décembre) si l'argument est une date

$_1$ NOMBRE ← **AN(** NOMBRE **)**

renvoie l'année si l'argument est une date

Fonctions auto-référentes

Il s'agit de fonctions à résultat numérique, qui s'appuient sur une variable numérique existante et en tirent une transformation. Ces fonctions nécessitent des calculs préalables à l'exécution sujet par sujet. Pour cette raison, elles sont de priorité 0, parce qu'elles doivent intercepter le nom de la variable avant qu'elle soit interprétée comme le type de valeur qu'elle représente :

$_0$ NOMBRE ← **Z(** CONSTANTE_TEXTUELLE **)**

La constante textuelle doit être le nom d'une variable numérique existante dans le corpus courant. Z est le nom que les statisticiens donnent traditionnellement à la *transformation centrée-réduite*, qui consiste à remplacer toute valeur x par $z = (x - m)/s$, où m est la moyenne de la distribution des x et s son écart-type. Le résultat est centré par l'opération $-m$, qui assure une moyenne nulle, et réduit par l'opération $/s$, qui assure un écart-type égal à 1. Le résultat est un nombre réel positif

ou négatif, dont le nombre de décimales à conserver sera réglé au niveau de l'en-tête de la formule (voir ci-après) ; il est prudent de prévoir des valeurs comprises entre –5 et +5, et on conserve généralement une décimale.

$_0$ NOMBRE ← **Z(** CONSTANTE_TEXTUELLE CONSTANTE_TEXTUELLE **)**

C'est la transformation *centrée-réduite selon catégories* : dans cette variante, la moyenne et l'écart-type utilisées pour la transformation ne sont pas calculées sur l'ensemble des sujets, mais séparément pour chacune des catégories, ou classes, définies par une variable nominale, ordinale ou logique dont le nom est la deuxième constante textuelle.

$_0$ NOMBRE ← **RANG(** CONSTANTE_TEXTUELLE **)**

Le rang est calculé selon les valeurs croissantes de la variable numérique dont l'intitulé est la constante textuelle, à partir de 1. Attention ! Prévoir que le maximum est à peu près égal au nombre de sujets, et qu'un rang peut avoir 0,5 comme partie décimale, à cause des *ex-ae*quo.

$_0$ NOMBRE ← **QUANT(** CONSTANTE_TEXTUELLE CONSTANTE_NUMERIQUE **)**

$_0$ NOMBRE ← **NORM(** CONSTANTE_TEXTUELLE CONSTANTE_NUMERIQUE **)**

Ces deux fonctions ont en commun de fournir un résultat entier, compris entre 0 et la constante numérique – 1. QUANT est la transformation en *quantiles*, qui s'efforce de répartir les sujets, en conservant leur ordre, dans des classes d'effectifs approximativement égaux, tandis que NORM, transformation de *normalisation*, s'attache à obtenir des classes aux effectifs approximativement proportionnels à ceux de la Loi Normale.

$_0$ LOGIQUE ← **DICHO(** CONSTANTE_TEXTUELLE **)**

Cette fonction donne un résultat logique, qui, pour chaque sujet, est VRAI si la valeur qu'a le sujet à la variable numérique (dont le nom est la constante textuelle) est supérieure à la médiane de la distribution, FAUX dans le cas opposé. Si on préfère pour résultat des valeurs numériques 0 et 1, on utilise plutôt QUANT(CONSTANTE_TEXTUELLE 2) ou encore on utilise la conversion NOMBRE(DICHO(CONSTANTE_TEXTUELLE)).

Comparaisons et prédicats

Un prédicat est la partie *droite* d'une comparaison : dans l'expression *a* = *b*, le prédicat est = *b*. Les prédicats sont principalement (mais pas seulement) destinés à construire des comparaisons dont le résultat est VRAI ou FAUX, à condition de confronter une valeur d'un certain type au prédicat approprié :

$_6$ LOGIQUE ← NOMBRE PREDICAT_NUMERIQUE

$_6$ LOGIQUE ← TEXTE PREDICAT_TEXTUEL

$_6$ LOGIQUE ← LOGIQUE PREDICAT_LOGIQUE

On obtient donc 3 types de prédicats, auxquels s'ajoute le prédicat d'existence, détaillé plus loin, qui sert à s'assurer si une valeur est ou non définie.

Prédicats numériques

$_6$ PREDICAT_NUMERIQUE ← **=** NOMBRE

Un tel prédicat ne fait rien tout seul : il aura besoin qu'y soit confronté un nombre en partie gauche. A la place de **=**, on peut trouver aussi **<**, **>**, **<=** (plus petit ou égal) **>=** (plus grand ou égal) et **<>** (différent).

Un prédicat numérique peut aussi être relatif à un intervalle :

$_6$ PREDICAT_NUMERIQUE ← **IN [** NOMBRE NOMBRE **]**

Ci-dessus, avec les bornes incluses. Les variantes [[,]] et] [sont également possibles.

Il peut enfin s'agir de la présence d'un nombre dans une série de nombres, ici notée NOMBRES. Les séries seront définies un peu plus loin.

$_6$ PREDICAT_NUMERIQUE ← **IN** NOMBRES

Prédicats logiques

Les possibilités sont assez restreintes :

6 PREDICAT_LOGIQUE ← **=** LOGIQUE

6 PREDICAT_LOGIQUE ← **<>** LOGIQUE

Prédicats textuels

Ils sont plus variés encore que les prédicats numériques :

6 PREDICAT_TEXTUEL ← **=** TEXTE

6 PREDICAT_TEXTUEL ← **<>** TEXTE

L'identité textuelle est évidente, de même que la non-identité. En revanche les inégalités relèvent d'une interprétation particulière :

6 PREDICAT_TEXTUEL ← **<** TEXTE

Il ne s'agit pas de longueur des textes, mais de *précédence* dans l'ordre lexicographique (on se méfiera toutefois des caractères accentués, dont le comportement peut dépendre de la version du système d'exploitation). On dispose aussi de **>**, **<=** et **>=**. Enfin, la présence d'un texte dans une série de textes (voir ci-après) peut être testée par :

6 PREDICAT_TEXTUEL ← **IN** TEXTES

Toutes ces comparaisons sont effectuées en tenant compte de la *casse*, c'est-à-dire de l'opposition majuscules/minuscules : un « A » n'est pas un « a ». Cependant, chacun de ces opérateurs de prédicats textuels possède une variante à *casse ignorée* (*case unsensitive*) qui s'en distingue par une apostrophe ajoutée :

6 PREDICAT_TEXTUEL ← **='** TEXTE

Ainsi,

```
Machin = MACHIN
```

est FAUX, tandis que

```
Machin =' MACHIN
```

est VRAI. On aura donc **>'**, **<' , <=', >=', <>'** et **IN'.**

Prédicats d'existence

6 PREDICAT_EXISTENCE ← **= RIEN**

6 PREDICAT_EXISTENCE ← **<> RIEN**

Le premier est VRAI après une valeur définie, le second, après une valeur indéfinie.

6 LOGIQUE ← NOMBRE PREDICAT_EXISTENCE

Ces prédicats s'appliquent également aux valeurs TEXTE et LOGIQUE.

Séries

Une série est une entité qui rassemble des valeurs du même type : il existe donc des séries de nombres, dont le type est NOMBRES, des séries de textes, dont le type est TEXTES, et des séries de logiques, dont le type est LOGIQUES. Noter le S final qui est la marque de la série.

Il existe trois sortes de séries : explicites, issues d'une collection et issues d'une collecte dans un système de base de données relationnelles. Ces dernières seront présentées dans une rubrique spéciale.

Séries explicites

Une série implicite est composée d'une suite de plusieurs valeurs de même type, encadrée d'une paire d'accolades. Une manière classique d'exposer la règle serait la suivante :

7 NOMBRES ← **{** NOMBRE … NOMBRE **}**

Mais un usager qui observe les commentaires en dessous de l'éditeur de formules constatera que plusieurs règles concourent en fait à cette construction :

7 DEBUT_NOMBRES ← **{** NOMBRE

7 DEBUT_NOMBRES ← **{** DEBUT_NOMBRES NOMBRE

7 NOMBRES ← **{** DEBUT_NOMBRES **}**

Ce mécanisme simple permet de construire des séries de n'importe quelle longueur. Les deux autres types ont aussi leurs séries :

7 LOGIQUES ← **{** LOGIQUE … LOGIQUE **}**

7 TEXTES ← **{** TEXTE … TEXTE **}**

Une série explicite peut-être étendue à gauche ou à droite par l'ajout d'un élément :

7 NOMBRES ← NOMBRES **+** NOMBRE

7 NOMBRES ← NOMBRE **+** NOMBRES

Et deux séries de même type peuvent être concaténées en une seule :

7 NOMBRES ← NOMBRES **+** NOMBRES

Ces fonctionnalités sont évidemment partagées par les autres types de séries explicites.

Référence à une collection

Une autre manière de créer une série est d'invoquer une collection : la série prend un type qui correspond à celui de la collection :

- TEXTES pour une collection de nominales ou d'ordinales
- LOGIQUES pour une collection de logiques
- NOMBRES pour une collection de numériques ou de calendaires

Le mécanisme profond qui identifie dans une CONSTANTE_TEXTUELLE une référence à collection plutôt qu'un texte quelconque est le même que celui qui décèle les références aux variables, raison de plus pour ne pas donner à une collection un nom que porte déjà une variable.

Les séries issues des collections acceptent, comme les séries explicites, extension et concaténation.

Sommations

Ce vocable est pris ici dans le sens inhabituel de « tout ce qui ressemble de près ou de loin à une somme sur une série de nombres ». Le modèle est celui-ci :

8 NOMBRE ← **SOMME** NOMBRES

C'est en effet l'usage principal des séries que d'effectuer sur elles des opérations collectives telles que la somme de leurs éléments.

Sommation sur séries de nombres

Dans le même esprit, on peut calculer sur une série de nombres leur moyenne, leur variance, leur maximum et leur minimum :

8 NOMBRE ← **MOYENNE** NOMBRES

8 NOMBRE ← **VARIANCE** NOMBRES

8 NOMBRE ← **MIN** NOMBRES

8 NOMBRE ← **MAX** NOMBRES

Le problème se pose alors de la contagion des valeurs indéfinies. Si un élément d'une somme est indéfini, la somme elle-même n'est-elle pas indéfinie ? En effet. Mais la difficulté peut être tournée avec des variantes des sommations :

8 NOMBRE ← **SOMME'** NOMBRES

Cette variante fait la somme des seuls éléments définis dans la série, en ignorant ceux qui ne le sont pas. On trouve aussi les variantes **MOYENNE', VARIANCE', MIN' et MAX'**

Sommation sur séries logiques

Comme la SOMME est une extension de l'addition, les sommations logiques sont des extensions des opérateurs ET et OU :

- LOGIQUE ← **INTERSECTION** LOGIQUES

Cette sommation est vraie si tous ses éléments sont vrais. Pour cette raison, elle possède un alias ET/.

- LOGIQUE ← **UNION** LOGIQUES

Cette sommation est vraie dès qu'un élément est vrai. Pour cette raison, elle possède un alias OU/. En raison de la logique ternaire exposée plus haut, il n'existe pas de variante aux sommations logiques qui ignorerait les valeurs indéfinies : pour le OU/ c'est implicite, pour le ET/ ce n'est pas souhaitable.

Sommation sur séries de textes

- TEXTE ← **CHAINE** TEXTES
- TEXTE ← **CHAINE_** TEXTES

Ces sommations concatènent les textes définis de la série, en ignorant les valeurs absentes. La seconder variante insère des espaces entre les textes concaténés.

Sommation sur toutes sortes de séries

Une opération basique commune à toutes les séries est de compter leur nombre d'éléments :

- NOMBRE ← **CARDINAL** NOMBRES
- NOMBRE ← **CARDINAL'** NOMBRES

La première version compte tous les éléments, définis ou pas, la seconde ne compte que les éléments définis. Cela existe aussi, bien sur, pour les TEXTES et LOGIQUES.

L'autre manœuvre commune à toutes les séries et de compter combien de leurs éléments satisfont un prédicat :

- NOMBRE ← **COMBIEN** NOMBRES PREDICAT_NUMERIQUE

Les valeurs indéfinies sont réputées ne pas satisfaire le prédicat. On trouvera la même chose pour les TEXTES avec bien sûr un PREDICAT_TEXTUEL, et les LOGIQUES avec un PREDICAT_LOGIQUE.

- NOMBRE ← **COMBIEN** NOMBRES PREDICAT_EXISTENCE

Ce comptage sur prédicats est possible sur toutes les sortes de séries. On remarquera que quelque chose comme :

```
COMBIEN série <> RIEN
```

est identique à `CARDINAL' série`.

De telles redondances sont assez fréquentes : il y a souvent plusieurs manières d'obtenir ce que l'on souhaite. La meilleure est celle que l'on comprend le mieux.

Cette règle est plutôt une règle de prélèvement que de sommation, mais, agissant sur les séries, elle trouve discrètement sa place ici.

- NOMBRE ← **NIEME** NOMBRE NOMBRES
- TEXTE ← **NIEME** NOMBRE TEXTES
- LOGIQUE ← **NIEME** NOMBRE LOGIQUES

extrait le $n^{ième}$ élément de la série, en partant de 0 pour le premier élément (sale manie d'informaticien, mais beaucoup de fonctions auto-référentes donnent aussi des résultats à partir

de 0). Le nombre n est d'abord arrondi ; s'il est négatif ou supérieur à l'indice du dernier élément de la série (sa longueur – 1), le résultat est non-défini.

Bases de données relationnelles

Les bases de données relationnelles sont une branche majeure de l'informatique de gestion, et engendrent à ce titre une imposante et complexe littérature, mais elles reposent en réalité sur des principes très simples, et Hector en propose une implémentation rustique mais suffisante.

Prenons, dans un collège, un fichier des élèves, et par ailleurs un fichier des classes. Pour pouvoir effectuer toutes sortes de calculs du côté des classes sur les élèves, et vice versa, il faut et suffit de savoir dans quelle classe se situe chaque élève. Dans un système de base de données classique, la fiche « élève » comporterait un champ « classe », qui, pour l'élève Adélaïde Macheprot, par exemple, contiendrait « 3ème2 ». Ce champ « classe » serait décrit comme *clef externe* vers le fichier des classes.

Les *variables* jouent dans Hector exactement le rôle des *champs* dans les bases de données. D'ailleurs un corpus Hector est une table de base de données. On pourrait donc imaginer que dans le corpus des élèves, il y ait une variable nominale « Classe », et que le sujet correspondant à Adélaïde y aurait la valeur « 3ème2 ». Cependant, pour des raisons d'efficacité (Hector est capable de traiter à grande vitesse des corpus volumineux), Hector n'accepte comme clef externe que des variables numériques positives et entières, qui représentent en fait des numéros *implicites* de sujets dans une autre base.

Ainsi, la variable « Classe » du corpus « élève » ne sera pas nominale mais numérique, et contiendra par exemple la valeur 14, étant entendu que le sujet 14 du corpus « Classes » désigne précisément la 3ème2. Le lien entre les deux tables est donc désormais assuré par la variable « Classe ». Ce lien est virtuel, et n'est activé que quand on en a besoin.

On peut en avoir besoin de deux manières. Appelons corpus-père le corpus le plus élevé hiérarchiquement, ici le corpus Classes, et corpus-fils l'autre, ici le corpus Elèves. *La variable qui fait le lien est toujours située dans le corpus-fils.* Si on souhaite récupérer dans le corpus-père des informations montant du corpus-fils, on fait ce qu'on appelle une *Collecte*. Si on souhaite récupérer dans le corpus-fils des informations descendant du corpus-père, on procède à un *Héritage*.

Collecte

Exemple de *collecte* : le corpus des classes s'appelle classes2008.cn, celui des élèves élèves2008.cn. Le corpus des élèves contient une clef externe vers les classes qui est la variable Classe, et, parmi d'autres, une variable numérique « Anglais trimestre 1 » qui dit bien ce que ça veut dire. On souhaite obtenir dans le corpus Classes une nouvelle variable contenant, pour chaque classe, la moyenne en anglais au premier trimestre. On utilise la règle de collecte sur les nombres (il y a bien sûr la même sur les textes et sur les logiques) :

0 NOMBRES ← **COLLECTE** CONSTANTE_TEXTUELLE **DE** CONSTANTE_TEXTUELLE **PAR** CONSTANTE_TEXTUELLE

La première constante textuelle est le nom de la variable à collecter, la seconde le nom du corpus, la troisième le nom de la variable-clef, celle qui fait le lien.

Dans notre exemple, ça donnerait :

```
MOYENNE( COLLECTE Anglais_trimestre_1 DE élèves2008 PAR Classe )
```

le rangement dans la variable « Anglais trimestre 1 » étant l'affaire de l'en-tête de la formule (voir ci-après). Cette formule est à exécuter dans le corpus-père, là où sera créée la nouvelle variable.

On remarque les _ à la place des espaces dans les noms de variable. Le nom du corpus-fils n'a pas besoin de l'extension .cn, qui sera rajoutée automatiquement ; le fichier est ici réputé placé dans le même répertoire que le corpus-père, faute de quoi il faut indiquer le chemin complet.

Le résultat de la collecte est une série. Dans la règle, on a écrit NOMBRES, mais en fait le type de série dépend de celui de la variable collectée. Les séries résultant de la collecte sont utilisables, comme les autres séries, avec tous les types de sommations, mais comme les valeurs constituant la série sont calculées « au vol » pour chaque sujet, les séries en provenance d'une collecte ne peuvent pas faire l'objet d'une concaténation ni d'une extension à droite ou à gauche : au niveau de l'analyse syntaxique, ce sera accepté, mais les éléments ajoutés seront ignorés à l'exécution. On peut néanmoins s'en tirer en sommant en plusieurs fois : une fois sur la série collectée, dont on réinjecte le résultat dans une autre série, par exemple explicite, sur laquelle on somme à nouveau.

Autre exemple : on veut savoir combien d'élèves de chaque classe on obtenu au moins dix au brevet blanc :

```
COMBIEN( COLLECTE note_brevet_blanc DE élèves2008 PAR Classe >= 10 )
```

On peut mettre de COLLECTE à Classe entre parenthèses pour faciliter la relecture si on en éprouve le besoin.

Héritage

Exemple d'héritage : le corpus Classes contient une variable « prof. de français ». Pour construire dans le corpus Elèves une variable logique qui est vraie si le prof de français de l'élève est M. Fruchard, on utilise la règle d'héritage :

0 NOMBRE ← **HERITE** CONSTANTE_TEXTUELLE **DE** CONSTANTE_TEXTUELLE
PAR CONSTANTE_TEXTUELLE

Là aussi, le résultat est NOMBRE si la première constante textuelle est le nom d'une variable numérique, mais ce peut-être aussi TEXTE ou LOGIQUE

```
HERITE prof_de_français DE classes2008 PAR Classe = Fruchard
```

Si on trouve ça peu clair, on peut mettre de HERITE à Classe entre parenthèses : ça ne mange pas de pain. La formule d'héritage est toujours dans le corpus-fils

Bases relationnelles multiples

Le lien qui a été décrit est toujours entre un corpus-père et un corpus-fils. Ce lien, en analyse classique de base de données, est appelé 1-n, parce qu'à un sujet du corpus-père correspondent plusieurs sujets du corpus-fils.

Or la théorie des bases de données relationnelles prévoit trois types de liens : 1-n, mais aussi 1-1 et surtout n-n.

Le lien 1-1 est facile à réaliser : c'est un cas particulier de 1-n. Il est assez peu utile puisque cela équivaut à avoir davantage de variables dans le même corpus. En fait il vient du souci de fractionner une même base en plusieurs, pour les systèmes de base de données qui ont besoin d'avoir toutes leurs variables en mémoire en même temps (ce n'est pas le cas d'Hector, qui ne charge que ce dont il a besoin quand il en a besoin) ; de plus, il soulève des problèmes d'intégrité relationnelle : quand on ajoute ou enlève un sujet d'un corpus, il faut faire de même dans les autres, etc …

Le lien n-n est le plus complexe conceptuellement : c'est par exemple le lien entre la base des moniteurs d'une auto-école et la base des apprentis-conducteurs : la rencontre peut se produire plusieurs fois, et pas toujours entre les mêmes.

En fait la complexité n'est que conceptuelle et cède à l'analyse : on aura en pratique trois corpus :

- Un corpus Moniteurs
- Un corpus Apprentis
- Un corpus Leçons

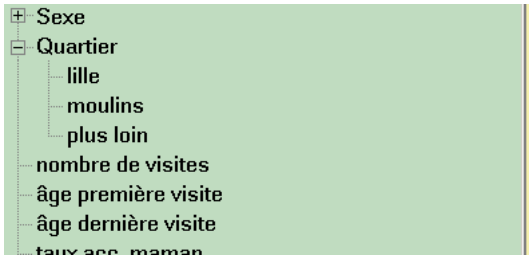
Il y aura un lien 1-n entre le corpus Leçons et le corpus Moniteurs, et un autre lien 1-n entre le corpus Leçons et le corpus Apprentis. En d'autres termes, le lien n-n n'a qu'une existence théorique, à un certain stade de l'analyse. Il est toujours décomposé en deux liens 1-n vers une

base intersection, parce qu'il n'y a pas moyen de faire autrement. Parfois, cette décomposition n'est pas évidente, et c'est pour ça qu'il y a des gens qui gagnent leur vie à le faire.

Outils d'accès aux corpus relationnels dans la page CORPUS

Le bouton [Chercher] donne accès à un dialogue d'ouverture de fichiers qui permet de désigner le corpus qu'on souhaite évoquer dans une formule relationnelle. Le nom du corpus apparaît dans la fenêtre verte, où on peut le cliquer pour l'insérer dans la formule en cours d'édition.

Le panneau des mots-clefs du langage est maintenant recouvert de la liste des variables du corpus relationnel, liste que l'on peut Cacher (pour accéder aux mots-clefs) ou Montrer avec les boutons correspondants.



Les variables ornées d'un + sont celles qui ont des étiquettes (nominales et ordinales). Comme le clic sert à ouvrir ou fermer ces sous-listes d'étiquettes, c'est ici le double-clic qui permet d'insérer un nom de variable ou d'étiquette dans la formule, toujours dans le but de ne pas risquer l'erreur de transcription, selon le précepte : « N'écris jamais toi-même ce que tu peux cliquer dans une liste ».

Structures conditionnelles

Les structures conditionnelles sont des moyens de choisir entre plusieurs valeurs selon une condition, exprimée par une valeur logique.

9 NOMBRE ← **SI** LOGIQUE **ALORS** NOMBRE **SINON** NOMBRE

Selon que la valeur LOGIQUE est vraie ou fausse, c'est la première ou la seconde valeur NOMBRE qui devient le résultat. Un fonctionnement identique existe pour TEXTE et LOGIQUE.

On peut parfois souhaiter affecter une valeur indéfinie à un résultat conditionnel :

9 NOMBRE ← **SI** LOGIQUE **ALORS** NOMBRE **SINON RIEN**

9 NOMBRE ← **SI** LOGIQUE **ALORS RIEN SINON** NOMBRE

RIEN peut-être remplacé par son alias @.

Qu'est-ce qu'une formule correcte ?

On en a pratiquement fini avec les éléments, il s'agit maintenant de considérer les formules complètes, et les conditions pour qu'elles soient correctes. Une formule correcte est un objet du type FORMULE. Il peut provenir de diverses constructions :

0 FORMULE ← FORMULE_CALENDAIRE

0 FORMULE ← FORMULE_NUMERIQUE

0 FORMULE ← FORMULE_LOGIQUE

0 FORMULE ← FORMULE_ORDINALE

0 FORMULE ← FORMULE_NOMINALE

Bien que les valeurs manipulées dans les formules soient seulement de trois genres fondamentaux[15], on voit resurgir ici les six types de variables[16], qui commanderont leur comportement pour les tris et les statistiques.

Calendaires

De manière générale, et comme bien d'autres textes, une formule se décompose en une en-tête et un corps :

- FORMULE_CALENDAIRE ← ENTETE_CALENDAIRE CORPS_NUMERIQUE
- ENTETE_CALENDAIRE ← **µ** CONSTANTE_TEXTUELLE
- CORPS_NUMERIQUE ← **:** NOMBRE **;**

En effet, au niveau du calcul, une variable calendaire est un nombre, en fait un nombre de jours écoulé depuis le 31 Décembre 1899, ou quelque chose de ce genre, les dates antérieures étant représentées par des nombre négatifs.

Ce n'est qu'au moment de l'affichage qu'un variable calendaire prend l'allure plus familière d'une date, du genre 14/07/1789. Si le NOMBRE possède des décimales, il est préalablement arrondi à l'entier.

La constante textuelle de l'entête est le nom de la future variable. Ce nom ne doit être porté par aucune variable d'origine ou dérivée (s'il en existe encore) ni par aucune collection[17]. Ce peut être le nom d'une variable formulée existante du même type, ce qui veut dire qu'on peut rejouer les variables formulées, soit pour les corriger, soit parce que de nouvelles données ont été saisies.

Il aurait pu sembler plus concis de décrire la formule calendaire ainsi :

- FORMULE_CALENDAIRE ← **µ** CONSTANTE_TEXTUELLE **:** NOMBRE **;**

C'est en effet plus commode tant que tout va bien[18]. Cependant, quand on se trompe, on a besoin de tous les détails pour comprendre le commentaire d'Hector et agir en conséquence.

Numériques

- FORMULE_NUMERIQUE ← ENTETE_NUMERIQUE CORPS_NUMERIQUE
- ENTETE_NUMERIQUE ← **#** CONSTANTE_TEXTUELLE

Des éléments facultatifs, mais souvent indispensables, les *paramètres*, peuvent compléter l'entête numérique :

- ENTETE_NUMERIQUE ← ENTETE_NUMERIQUE MINIMUM_NUMERIQUE
- ENTETE_NUMERIQUE ← ENTETE_NUMERIQUE MAXIMUM_NUMERIQUE
- ENTETE_NUMERIQUE ← ENTETE_NUMERIQUE DECIMALE_NUMERIQUE
- MINIMUM_NUMERIQUE ← **MINIM** CONSTANTE_NUMERIQUE
- MAXIMUM_NUMERIQUE ← **MAXIM** CONSTANTE_NUMERIQUE
- DECIMALE_NUMERIQUE ← **DECIM** CONSTANTE_NUMERIQUE

MINIM et MAXIM fixent le maximum et le minimum des valeurs acceptables, et DECIM le nombre de décimales.

Les valeurs qui, telles que calculées par la formule, seraient extérieures aux minimum/maximum *ne sont pas codées dans la nouvelle variable* : elles sont remplacées par des non-valeurs. Il convient dont d'être attentif à ce que l'on fait : par défaut, autrement dit si on n'indique explicitement aucun

---

[15] Textes, Nombres, Logiques

[16] Calendaire, Numérique, Logique, Ordinale, Nominale, Texte-libre

[17] Ni par une étiquette dans une variable qui pourrait être employée dans une autre formule, car à cause de sa stratégie de préemption, Hector privilégie toujours l'hypothèse qu'il a affaire à une référence à une variable ou à une collection, si le nom le permet.

[18] Le retour à la ligne est pour Hector la même chose qu'un espace : un simple séparateur.

paramètre, le minimum est de 0, le maximum de 100, et le nombre de décimales de 0. C'est parfait tant qu'on travaille sur des pourcentages entiers, mais si on veut calculer des rangs, il vaut mieux se souvenir que certains rangs sont du genre 13,5, et si on calcule une transformation centrée-réduite, elle va couramment de –5 à 5 et qu'il est d'usage de lui laisser au moins une décimale.

Pour ces raisons, il est souvent utile, après exécution d'une variable formulée, d'aller voir en page TRAITEMENT l'allure qu'elle a quand on la trie, et notamment combien de sujets sont pris en compte, puisque les non-valeurs ne le sont pas. La disparition inopinée de nombreux sujets dans une variable a souvent pour cause une éviction par des paramètres mal fixés. Croiser une variable formulée avec sa ou ses variables-sources (celles qui sont évoquées dans la formule) est parfois riche d'enseignements.

Logiques

- FORMULE_LOGIQUE ← ENTETE_LOGIQUE CORPS_LOGIQUE
- ENTETE_LOGIQUE ← **£** CONSTANTE_TEXTUELLE
- CORPS_LOGIQUE ← **:** LOGIQUE **;**

On le voit, la formule logique est assez dépourvue de mystère : comme elle prend nécessairement ses valeurs dans le couple VRAI / FAUX, elle n'a pas besoin de paramètres.

Ordinales

- FORMULE_ORDINALE ← ENTETE_ORDINALE CORPS_TEXTUEL
- ENTETE_ORDINALE ← **§** CONSTANTE_TEXTUELLE ETIQUETTES
- CORPS_TEXTUEL ← **:** TEXTE **;**
- DEBUT_ETIQUETTES ← **ETIQ{** CONSTANTE_TEXTUELLE
- DEBUT_ETIQUETTES ← DEBUT_ETIQUETTES CONSTANTE_TEXTUELLE
- ETIQUETTES ← DEBUT_ETIQUETTES **}**

Ce qui caractérise une variable ordinale, c'est de posséder des valeurs du genre texte parmi lesquelles un ordre strict est défini. De ce fait, le paramètre ETIQUETTES est indispensable, car il fixe l'ordre des étiquettes, qui doivent être des textes d'une longueur maximale de 10 caractères, y compris les espaces, qui y sont d'ailleurs représentés par des _.

L'existence d'étiquettes explicitement définies autorise l'utilisation de la fonction N°(, qui renvoie l'étiquette placée au rang passé en argument de cette fonction, rang compté à partir de 0. Ainsi, après `ETIQ{ a b c d }`, l'expression `N°( 1 )` désigne l'étiquette « b ».

Nominales

Les nominales sont aussi des variables à étiquettes, mais sans ordre.

- FORMULE_NOMINALE ← ENTETE_ NOMINALE CORPS_ TEXTUEL
- ENTETE_NOMINALE ← **$** CONSTANTE_TEXTUELLE
- ENTETE_NOMINALE ← ENTETE_NOMINALE ETIQUETTES

La dernière formule signale que la présence d'étiquettes dans l'entête nominale est facultative : le jeu d'étiquettes peut aussi bien être fabriqué « au vol », mais on préférera souvent l'indiquer explicitement pour garantir l'ordre de présentation, qui, même s'il n'a pas la même signification que pour une variable ordinale, ne peut quand même pas toujours être laissé au hasard. De plus, l'existence d'étiquettes explicites autorise l'emploi de la fonction N°(.

Libres

Les variables « texte libre » sont des variables à valeurs du genre TEXTE, sans limitation de longueur et sans opérations statistiques possibles.

- FORMULE_NOMINALE ← ENTETE_ LIBRE CORPS_ TEXTUEL
- ENTETE_NOMINALE ← **¤** CONSTANTE_TEXTUELLE

L'usage principal des variables textes libres est de conserver dans le même corpus des variables illustratives telles que le nom et le prénom, mais on peut aussi s'en servir pour récupérer des données venant de l'extérieur et les analyser avec des formules appropriées. Elles servent aussi à coder les séquences (version professionnelle/recherche).

Formules multiples et sur collections

Multiples

L'éditeur de formules peut contenir plusieurs formules successives, définissant plusieurs variables. Chaque variable formulée gardera le texte de *sa* propre définition, mais le contenu de l'éditeur de formules peut être sauvegardé dans un seul fichier .fch (formules de calcul d'Hector), ce qui permet, après une modification de données, de faire rejouer d'un seul coup plusieurs variables formulées.

De ce fait l'élément attendu sous la LISTE GENERALE pour attester d'une ou plusieurs formules correctes et activer le bouton [Exécuter] n'est pas FORMULE mais FORMULES.

Comme souvent on n'en fait qu'une à la fois, FORMULE suffit pour définir FORMULES :

0 FORMULES ← FORMULE

Mais on peut en avoir plusieurs[19] :

0 FORMULES ← FORMULES FORMULE

Collections

Les collections sont, rappelons-le, des ensembles de variables du même type. On peut souhaiter appliquer une même formule à toute une collection. Le moyen d'y parvenir est le suivant :

0 TEXTE_EXECUTABLE ← \ .....

0 FORMULES ← ENTETE_POUR TEXTE_EXECUTABLE

0 ENTETE_POUR ← **POUR** CONSTANTE_TEXTUELLE **+** CONSTANTE_TEXTUELLE

Tout ce qui dans l'éditeur est placé après le \ est considéré comme du texte exécutable.

L'une des constantes textuelles est le nom de la collection, l'autre est un affixe pour les nouvelles variables. Selon qu'un nom de collection est reconnu en première ou en seconde position, l'affixe sera un suffixe ou un préfixe.

Cet affixe viendra s'accoler devant ou derrière le nom de chacune des variables de la collection d'origine pour former le nom de la nouvelle variable.

Le texte exécutable de la formule obéit à la même syntaxe que les autres formules, sauf qu'au lieu d'une CONSTANTE_TEXTUELLE pour le nom de la nouvelle variable on utilisera le mot-clef NOUVELLE, et au lieu de la CONSTANTE_TEXTUELLE désignant l'une des variables de la collection, on utilisera le mot-clef SOURCE.

Dès lors le mécanisme est assez simple : pour chacune des variables de la collection, son intitulé vient remplacer le mot SOURCE dans le TEXTE_EXECUTABLE, tandis que le même intitulé augmenté de l'affixe remplace le mot NOUVELLE, après quoi le TEXTE_EXECUTABLE est exécuté comme une formule ordinaire.

Toutefois, comme le texte de la formule change à chaque tour, la vérification syntaxique n'est faite qu'au moment de l'exécution du POUR. On teste donc d'abord généralement la formule sur une seule variable pour vérifier sa correction syntaxique, avant de l'inclure dans une structure POUR.

[19] Et, comme cette règle est récursive, un nombre quelconque de FORMULE peuvent s'ajouter pour former un élément FORMULES.

Pour fixer les idées, à supposer qu'on dispose d'une collection numérique appelée Notes et contenant les variables Anglais, Français, Maths, Histoire, la formule

```
POUR rang_en_ + Notes \
# NOUVELLE DECIM 1
: RANG( SOURCE ) ;
```

provoque la création de quatre nouvelles variables « rang en Anglais », « rang en Français », « rang en Maths » et « rang en Histoire », qui sont les rangs de classement dans ces différentes matières (sous l'hypothèse qu'il n'y a pas plus de 100 sujets concernés ; sinon, augmenter le maximum par un paramètre MAXIM).

Chaque variable ainsi créée possède sa propre formule. Ainsi rang en Anglais a pour formule (qui peut être relue individuellement) :

```
# rang_en_Anglais DECIM 1
: RANG( Anglais ) ;
```

Rappelons que le paramètre DECIM 1, qui autorise un chiffre après la virgule, est nécessaire à cause des ex-aequo, qui peuvent générer des rangs du type 4,5.

LE FONCTIONNEMENT DU LANGAGE : EXEMPLES COMMENTES

Une des caractéristiques principales de ce langage, qui fait toute sa force, est la récursivité. La définition de ce concept est la suivante : est récursif tout élément qui peut figurer aussi bien dans la partie droite que dans la partie gauche d'une règle de syntaxe. Avec ça vous êtes bien avancés ! Concrètement, cela veut dire que les différentes structures peuvent être indéfiniment emboîtées les unes dans les autres, comme les poupées russes, sauf qu'il n'y a pas de règle sur le nombre de poupées. Bien sûr ce nombre ne peut être infini, et la limite concrète d'une formule n'est pas dans sa profondeur de récursion, mais dans le nombre de variables-source qu'elle peut évoquer simultanément : 1000, c'est dire qu'il y a de la marge.

Exemple de récursivité :

```
§ mention_en_méthodo
ETIQ{ échec P AB B TB }
: SI Note_en_Méthodologie < 10 ALORS échec
SINON SI Note_en_Méthodologie < 12 ALORS P
SINON SI Note_en_Méthodologie < 14 ALORS AB
SINON SI Note_en_Méthodologie < 16 ALORS B SINON TB ;
```

Pour essayer de comprendre comment raisonne, il faut identifier les règles qu'il applique

```
§ mention_en_méthodo
ETIQ{ échec P AB B TB }
```

est une authentique entête de formule ordinale, avec le signe §, un nouveau nom, et la définition d'une liste d'étiquettes autorisées : en effet, il s'agit des mentions, dont chacun se souvient avec bonheur ou malheur qu'elles sont strictement hiérarchisées. Sans trop entrer dans le détail, la remplacer par *EN_TETE_ORDINALE* est une application de la règle :

$_0$ ENTETE_ORDINALE ← **§** CONSTANTE_TEXTUELLE ETIQUETTES

A plusieurs reprises dans la formule, nous trouvons des expressions telles que :

```
Note_en_Méthodologie < 10
```

Elles se décomposent ainsi : Note en méthodologie est une numérique, < 10 est un prédicat numérique, le résultat est donc une logique.

Le mot échec, les mots P, AB, B, TB, sont des textes. Ceci nous permet des simplifications de la formule, où nous indiquons en italique les éléments créés par l'interprétation et la réduction d'éléments plus simples :

```
EN_TETE_ORDINALE
: SI LOGIQUE ALORS TEXTE
SINON SI LOGIQUE ALORS TEXTE
SINON SI LOGIQUE ALORS TEXTE
SINON SI LOGIQUE ALORS TEXTE SINON TEXTE ;
```

La dernière expression :

```
SI LOGIQUE ALORS TEXTE SINON TEXTE
```

appelle l'application de la règle

$_9$ TEXTE ← **SI** LOGIQUE **ALORS** TEXTE **SINON** TEXTE

elle peut donc être remplacée par TEXTE :

EN_TETE_ORDINALE
: SI *LOGIQUE* ALORS *TEXTE*
SINON SI *LOGIQUE* ALORS *TEXTE*
SINON SI *LOGIQUE* ALORS *TEXTE*
SINON *TEXTE* ;

Et c'est là qu'intervient la récursivité : cet élément texte qui provient d'une structure conditionnelle va à son tour être employé dans une autre structure conditionnelle. Ainsi, par trois fois encore, SI *LOGIQUE* ALORS *TEXTE* SINON *TEXTE* va être *réduit* en *TEXTE*.

Au bout du compte, la formule sera réduite à ceci :

EN_TETE_ORDINALE
: *TEXTE* ;

intervient alors la règle

$_0$ CORPS_TEXTUEL ← : TEXTE ;

reste :

EN_TETE_ORDINALE CORPS_TEXTUEL

et en appliquant :

$_0$ FORMULE_ORDINALE ← ENTETE_ORDINALE CORPS_TEXTUEL

FORMULE_ORDINALE

Puis, routine,

$_0$ FORMULE ← FORMULE_ORDINALE

$_0$ FORMULES ← FORMULE

et, le niveau *FORMULES* étant atteint, Hector est d'accord pour exécuter le texte contenu dans l'éditeur.

Est-ce que Hector raisonne *réellement* comme cela ? Hector ne raisonne pas, ce n'est qu'un programme, mais c'est bien cette logique-là qu'il applique.

Recherche d'erreurs

Et l'utilisateur a tout intérêt à y regarder de près, parce que c'est le moyen de détecter les erreurs de formulation.

En effet, qu'est-ce qu'une formule erronée ? C'est un texte qu'il n'a pas été possible de réduire à un élément *FORMULES*. Elle s'affiche donc dans le commentaire d'Hector en plusieurs morceaux, les coupures apparaissant là où la réduction n'a pas été possible, parce qu'aucune règle ne le permettait.

Pourquoi cela n'a-t-il pas été possible ? Parce qu'une réduction escomptée n'a pas eu lieu, à cause d'une erreur de vocabulaire ou de syntaxe. Cette réduction ratée bloque tout le processus récursif de réduction vers *FORMULES*.

Comment savoir où est l'erreur ? C'est très facile. Là où une réduction n'a pas été possible, au lieu d'une bloc homogène, la zone des commentaires d'Hector montre un *hiatus*, une interruption, une ligne blanche à l'endroit où il n'y a pas eu moyen de se mettre d'accord. Nous allons essayer d'explorer quelques types d'erreurs assez faciles à commettre dans un premier temps, à éviter ensuite.

Espace et orthographe

```
# total_maths
:algébre + géométrie ;
```

Elle avait l'air bien, pourtant cette petite formule ! Que dit ce bavard d'Hector ?

```
LISTE GENERALE ...
ENTETE NUMERIQUE ... # total_maths
TEXTE ... :algébre
CLEF ... +
NOMBRE ... géométrie
CLEF ... ;
```

Il a bien reconnu l' `ENTETE NUMERIQUE`, mais il nous dit `TEXTE ... :algébre`, alors qu'algèbre est une variable numérique, et donc un nombre ! Regardons mieux : nous avons collé le : et le mot algèbre, alors qu'ils doivent être séparés par un espace. Hélas oui, Hector est aussi pointilleux que ça ! Ce n'est qu'un logiciel, après tout.

On corrige :

```
ENTETE NUMERIQUE ... # total_maths
CLEF ... :
TEXTE ... algébre
CLEF ... +
NOMBRE ... géométrie
CLEF ... ;
```

Toujours pas content ! Et encore `TEXTE ... algébre` au lieu de le reconnaître comme un nombre. C'est que la variable s'appelle algèbre, avec un accent grave, et non algébre avec un accent aigu. Comment se prémunir contre de telles erreurs, si fréquentes ? Ne *jamais* écrire soi-même le nom de quelque chose qui existe déjà, comme un nom de variable, de collection, une étiquette ou un mot clef, mais aller le *cliquer* dans la liste où il figure : Hector l'écrira exactement comme il faut, en remplaçant les espaces par des _ et en ajoutant un espace à la fin pour ne pas coller au mot suivant. C'est vraiment une habitude à prendre tout de suite pour éviter de s'arracher les cheveux en se demandant pourquoi Hector refuse une formule alors qu'elle *a l'air* correcte.

Type des variables

```
# n'importe_quoi
: algèbre + Sexe ;
```

Qu'est-ce qui ne va pas dans cet exemple volontairement absurde ?

```
ENTETE NUMERIQUE ... # n'importe_quoi
CLEF ... :
NOMBRE ... algèbre
CLEF ... +
TEXTE ... Sexe
CLEF ... ;
```

Hector ne s'intéresse pas du tout au sens des mots, mais au type des variables, beaucoup : il ne possède aucune règle permettant de réduite NOMBRE + TEXTE. Autrement dit, l'addition est impossible entre ces éléments.

Il est toujours intéressant de lire ce que dit Hector quand ça ne marche pas :

```
# algèbre_selon
: Z( algèbre géométrie ) ;
```

Ca paraît sympa, non ?

```
ENTETE NUMERIQUE ... # algèbre_selon
CLEF ... :
ERREUR ... géométrie n'est pas le nom d'une variable catégorielle
CLEF ... ;
```

Mais ça ne marche pas : ici le message d'Hector est clair : « géométrie n'est pas le nom d'une variable catégorielle », autrement dit d'une variable créant des catégories, c'est-à-dire une nominale, une ordinale ou une logique.

```
# algèbre_selon
: Z( algèbre Sexe ) ;
```

Là, ça marche : la nouvelle variable contiendra la transformée centrée-réduite de la note en algèbre, calculée séparément selon le sexe.

Problèmes classiques et moins classiques

On présente ici quelques unes des applications les plus fréquentes en matière de formules.

Calculer une moyenne sur une série

On suppose qu'on dispose d'une collection « matières », qui rassemble les variables « algèbre », « géométrie », « histoire », « français », lesquelles contiennent des notes sur 20.

```
# moyenne_des_matières
: MOYENNE matières ;
```

On a utilisé ici la collection pour former la série qui est l'argument de MOYENNE. On aurait pu tout aussi bien se fatiguer à décrire explicitement la série :

```
# moyenne_des_matières
: MOYENNE { algèbre géométrie histoire français } ;
```

Remarquons que, dans les deux cas, on n'a pas précisé de minimum ni de maximum ni de décimales. Pour les *extrema*, tout va bien avec les valeurs par défaut 0 .. 100. Sans décimale, la moyenne va être arrondie, ce n'est pas forcément ce qu'on souhaite. Sans doute fera-t-on plutôt ceci :

```
# moyenne_des_matières DECIM 2
: MOYENNE matières ;
```

A première vue la formule suivante serait équivalente :

```
# moyenne_des_matières
: SOMME { algèbre géométrie histoire français } / 4 ;
```

si on tient vraiment à faire tout le travail soi-même ! C'est pourtant l'occasion d'introduire une réflexion sur les valeurs absentes. Tout va bien dans les formules ci-dessus tant que chaque élève a effectivement une note dans chaque matière. Que se passe-t-il pour Adélaïde Chaprot, qui a eu une angine blanche le jour de la compo' d'algèbre, et qui n'a donc que trois notes ? Avec SOMME et MOYENNE, le règlement est strict : pas toutes les notes, pas de moyenne possible : le résultat est pour l'infortunée une non-valeur. Souvent, c'est ce qu'on voudra faire, mais parfois on voudra autre chose, par exemple s'il est normal que tout le monde n'ait pas le même nombre de notes (il y en a qui font javanais troisième langue facultative). Que faire ? On emploiera dans ce cas l'alternative « valeurs définies seulement » :

```
# moyenne_des_matières DECIM 2
: MOYENNE' matières ;
```

Bien sûr, ça n'a pas la même signification. Et il n'est plus question de diviser la somme par 4, puisqu'on ne sait pas combien de matières chacun a réellement. D'ailleurs, comment le savoir si on s'y intéresse ?

```
# nombre_de_notes_connues
: COMBIEN matières <> RIEN ;
```

Cette dernière formule, qui utilise un prédicat d'existence, introduit par ailleurs une réflexion intéressante :

Prédicats d'existence, paradoxes et variable-témoin

L'usage des prédicats d'existence pourrait théoriquement entraîner des paradoxes. Ainsi une variable servant à recenser les sujets dont l'âge est inconnu pourrait avoir l'allure suivante :

```
£ âge_inconnu
: âge = RIEN ;
```

Cette formule pourrait donner une valeur vraie à la variable « âge inconnu » à tous les *sujets virtuels*, c'est à dire tous les sujets qui n'existent pas parce qu'ils sont au delà du dernier sujet connu (ce qui serait assez facile à contrôler), ou, plus difficile à repérer, parce qu'ils correspondent à des trous dans la numérotation. En triant la variable logique créée par la formule, on aurait la surprise de voir afficher un bien plus grand nombre de sujets qu'on ne s'en connaissait. Pour empêcher cela, Hector utilise la notion de *variable-témoin*, c'est-à-dire de variable pour laquelle seuls les sujets existant réellement ont une valeur. Il faut choisir une variable dont on est sûr que tous les sujets y ont une valeur. La variable-témoin est la première variable dont le nom commence par @, ou, s'il n'en est aucune, la première variable du corpus ; on déplacera si besoin est une variable appropriée vers le premier rang dans l'ordre d'affichage.

Regrouper une variable nominale

On a une variable avec trop d'étiquettes et on voudrait les regrouper dans un système plus concis : ici, il s'agit de modalités d'accès à une licence :

```
$ Origine_regroupée
: SI Origine IN { D.Lettre D.Sces D.ScHum } ALORS deug
SINON SI Origine IN { Soc.Form Tech.Sup } ALORS autreBac+2
SINON validation ;
```

On notera l'usage du prédicat IN TEXTES et les SI en cascade.

L'expression

```
Origine IN { Soc.Form Tech.Sup }
```

équivaut à

```
( Origine = Soc.Form) ou ( Origine = Tech.Sup )
```

Ca paraît un peu plus concis, mais bien sûr c'est affaire de goût, et, encore une fois, la bonne formule est celle que l'on comprend.

On notera également qu'il n'a pas été nécessaire de préparer explicitement la liste d'étiquettes. La contrepartie est que l'ordre dans lequel elles seront affichées dépende de ce qui se présente en premier parmi les sujets.

Il faut savoir aussi que les sujets dont le cas n'a pas été traité dans la formule reçoivent une non-valeur. Il sera donc souvent utile de croiser la nouvelle variable avec l'ancienne pour vérifier qu'on n'a perdu personne.

Faire des classes d'âge

C'est quelque chose que l'on fait assez souvent. Il y a deux grandes approches : ou bien on veut des classes d'âge à intervalles réguliers, ou bien on veut des classes d'âge approximativement équiprobables (autant de sujets dans chaque classe).

Dans les deux cas, il s'agit d'ordinales, on définit donc au préalable les étiquettes.

Pour des intervalles réguliers, on aura quelque chose de ce genre :

```
§ Classes_d'âge_régulières
ETIQ{ 15-19ans 20-24ans 25-29ans 30-34ans }
: SI Age IN [ 15 19 ] ALORS 15-19ans
SINON SI Age IN [ 20 24 ] ALORS 20-24ans
SINON SI Age IN [ 25 29 ] ALORS 25-29ans
SINON SI Age IN [ 30 34 ] ALORS 30-34ans SINON RIEN ;
```

Il y a moyen de faire plus léger, mais en gros c'est l'esprit. Dans cet exemple, on a explicitement prévu de négliger les plus de 34 ans (arbitrairement, c'est pour l'exemple), mais les éventuels moins de 15 ans sont tout aussi exclus par la formule, puisque leur cas n'est jamais traité.

Dans l'autre cas, celui des classes équiprobables, on ne sait pas d'avance ce que seront les âges limites :

```
§ Classes_d'âge_équiprobables
ETIQ{ groupe0 groupe1 groupe2 groupe3 }
: groupe + TEXTE( QUANT( Age 4 ) ) ;
```

QUANT(Age 4), fonction quantile à quatre cases, autrement dit quartile, renvoie un nombre 0, 1, 2 ou 3. La fonction TEXTE( ) le transforme ensuite en un texte qui vient se coller au texte « groupe » pour former une étiquette convenable.

Une alternative possible, dans un autre style :

```
§ Classes_d'âge_équiprobables_2
ETIQ{ jeunes exJeunes mûrs vétérans }
: N°( QUANT( Age 4 ) ) ;
```

La fonction N°(prend pour argument le nombre de 0 à 3 fourni par QUANT(et l'exploite comme indice d'une étiquette (à partir de 0) dans la liste des étiquettes définies. Ainsi, `N°( 2 )` renvoie ici à l'étiquette « mûrs ».

Utiliser une formule sur collection

On dispose d'une collection « Matières » qui rassemble les notes dans quatre disciplines. On veut créer des variables logiques qui sont vraies si la barre des 10 (couramment et improprement appelée « moyenne ») a été atteinte ou franchie ; comme le raisonnement est le même dans tous les cas, autant tout faire en même temps :

```
POUR matières + _Ok \
£ NOUVELLE
: SOURCE >= 10 ;
```

L'exécution de cette formule crée quatre variables, chacune d'elles ayant sa propre formule :

```
£ algèbre_Ok
 : algèbre >= 10 ;
```

et de même pour géométrie, histoire, français.

Jouer avec les dates

A supposer que « la date » soit une variable calendaire, la formule suivante vous l'écrira en clair :

```
¤ date_en_clair
: CHAINE_ {
Le
NIEME DI( la_date ) { Lundi Mardi Mercredi Jeudi Vendredi Samedi }
TEXTE( JOUR( la_date ) )
NIEME MOIS( la_date ) { Janvier Février Mars Avril Mai Juin Juillet Août
Septembre Octobre Novembre Décembre }
TEXTE( AN( la_date ) )
} ;
```

Ainsi, si « la date » est le 12/03/2008, « la date en clair » sera « Le Mercredi 12 Mars 2008 ».

INVENTAIRE RECAPITULATIF DES MOTS-CLEFS

Eléments de structure

" outil de « textification impérative » du mot qui suit, qu'il s'agisse d'un texte, d'un nombre, d'un mot-clef ou d'une référence

: début d'un corps de formule

; fin d'un corps de formule

. séparateur décimal dans les nombres

, séparateur décimal alternatif

(parenthèse ouvrante

) parenthèse fermante : équilibre une parenthèse ouvrante pour « protéger » une expression à interprétation prioritaire, ou pour fermer une fonction du type XXX(

[crochet ouvrant

] crochet fermant. Les crochets sont utilisés exclusivement dans les prédicats d'intervalle numérique avec IN

En-têtes et paramètres

µ symbole d'entête calendaire

£ symbole d'entête logique

symbole d'entête numérique

MAXIM facultatif dans une entête numérique (valeur maximale)

MINIM facultatif dans une entête numérique (valeur minimale)

DECIM facultatif dans une entête numérique (nombre de décimales)

$ symbole d'entête nominale

§ symbole d'en tête ordinale

ETIQ{ constructeur de jeu explicite d'étiquettes, obligatoire pour les ordinales, facultatif pour les nominales. Doit être fermé par une accolade fermante

¤ symbole d'entête texte-libre

Opérateurs mathématiques

-- opposé d'un nombre (moins unaire)

+ addition des nombres, mais aussi concaténation des textes et des séries ainsi que séparateur du nom de collection et de l'affixe dans une structure POUR

+_ concaténation des textes avec insertion d'un espace

- soustraction

* multiplication

/ division

** exponentiation ou élévation à une puissance. Remarque : la racine carrée est l'élévation à la puissance ½, ou 0.5

² carré

Fonctions mathématiques

(nécessitent une parenthèse fermante)

ABS(	valeur absolue d'un nombre (suppression du signe)
TRONQUE(	suppression des décimales
ARR(	arrondi, soit à l'entier avec un seul argument, soit au nombre de décimales indiqué par le second argument
LN(	logarithme naturel ou népérien
LOG10(	logarithme en base 10
LOG2(	logarithme en base 2
EXP(	exponentielle
UNIFORME(	distribution aléatoire uniforme réelle entre deux bornes, la seconde exclue
NORMALE(	distribution normale selon moyenne et écart-type passés en arguments
ALEA(	distribution aléatoire uniforme entière entre bornes incluses

Manipulation de textes

TAILLE(	nombre de caractère d'un textes, espaces inclus
PRESENCE(	présence d'un sous-texte dans un texte
POSITION(	position d'un sous-texte dans un texte
OCCURRENCES(	nombre d'occurrences d'un sous-texte dans un texte
MAJUSCULES(	conversion d'un texte en majuscules
MINUSCULES(	conversion d'un texte en minuscules
COMPACTE(	remplace dans un texte xxxxxx par x6 (par exemple)
DECOMPACTE(	remplace dans un texte l'expression « x6 » par xxxxxx (par exemple)
PREMIERS(	extraction des n premiers caractères d'un texte
DERNIERS(	extraction des n derniers caractères d'un texte
RESUME(	résumé d'un texte en n caractères au plus
EXTRAIT(	extraction d'un sous-texte commençant à une position et sur une longueur passées en arguments
REMPLACE(	remplace un premier texte par un second dans un troisième

Divers et conversions

SUJET	renvoie le numéro d'ordre implicite du sujet en cours de traitement
TEXTE(	convertit en texte les valeurs logiques ou numériques
LOGIQUE(	convertit en logique les valeurs textuelles ou numériques
NOMBRE(	convertit en nombre les valeurs logiques ou textuelles
DATE(	interprète un texte au format jj/mm/aaaa comme une date
DI(	renvoie 0 (Lundi) à 6 (Dimanche) si l'argument est une date
JOUR(	renvoie le quantième du jour dans le mois si l'argument est une date
MOIS(	renvoie 0 (Janvier) à 11 (Décembre) si l'argument est une date
AN(	renvoie l'année si l'argument est une date
N°(	renvoie l'étiquette explicite dont le rang (à partir de 0) est passé en argument.

Fonctions auto-référentes

Z(	transformation centrée-réduite, éventuellement selon les valeurs d'une variable

catégorielle passée en second argument

RANG(rang du sujet dans une variable numérique ou ordinale

QUANT(transformation en n quantiles

DICHO(dichotomie logique sur la médiane

NORM(normalisation en n classes

Sommations et calcul sur séries

{ accolade ouvrante : début d'une série

} accolade fermante : conclut les séries et ETIQ{

toutes séries

CARDINAL compte le nombre d'éléments d'une série, définis ou non

CARDINAL' compte le nombre d'éléments définis d'une série

NIEME extrait le $n^{ième}$ élément de la série

numériques

MAX maximum d'une série de nombres définis

MAX' maximum des éléments définis d'une série numérique

MIN minimum d'une série de nombres définis

MIN' minimum des éléments définis d'une série numérique

COMBIEN nombre de fois où un prédicat est satisfait par les éléments d'une série

SOMME somme d'une série de nombres définis

SOMME' somme des éléments définis d'une série numérique

MOYENNE moyenne d'une série de nombres définis

MOYENNE' moyenne des éléments définis d'une série numérique

VARIANCE variance d'une série de nombres définis

VARIANCE' variance des éléments définis d'une série numérique

logiques

UNION OU étendu à une série logique

OU/ alias d'UNION

INTERSECTION ET étendu à une série logique

ET/ alias d' INTERSECTION

Textes

CHAINE concaténation d'une série de textes

CHAINE_ concaténation d'une série de textes avec insertion d'espace

Constantes et opérateurs logiques

VRAI constante vraie

FAUX constante fausse

NON négation d'une logique

ET intersection : vrai si et seulement si les opérandes sont vrais

OU union : vrai dès qu'un opérande est vrai

OUX ou exclusif : vrai si un seul des opérandes est vrai

Opérateurs de comparaison numérique et texte

= égal pour les nombres et logiques, identique pour les textes
< plus petit pour les nombres, précède alphabétiquement pour les textes
> plus grand pour les nombres, suit alphabétiquement pour les textes
<= plus petit ou égal, identique ou précède
>= plus grand ou égal, identique ou suit
<> différent
IN est inclus dans une série ou un intervalle (nombres)

Comparaisons de texte (casse ignorée : majuscule = minuscule)

=' égalité hors casse
<' précède hors casse
>' suit hors casse
<=' identique ou précède hors casse
>=' identique ou suit hors casse
<>' différent hors casse
IN' est inclus dans une série de textes, si on ne tient pas compte de la casse

Structures conditionnelles

SI introduit la structure SI condition ALORS valeur SINON valeur, structure qui doit obligatoirement être complète
ALORS introduit la valeur à affecter si la condition est vraie
SINON introduit la valeur à affecter si la condition est fausse
RIEN non-valeur, valeur indéfinie
@ alias de RIEN

Formules par collections

POUR introduit l'en-tête de POUR collection + suffixe \ *texte exécutable* ou POUR préfixe + collection \ *texte exécutable*
NOUVELLE représente la variable à créer dans un *texte exécutable*
SOURCE représente la variable source dans un *texte exécutable*

Bases relationnelles

COLLECTE introduit la formule de collecte COLLECTE *variable* DE *corpus* PAR *clef*
HERITE introduit la formule d'héritage HERITE *variable* DE *corpus* PAR *clef*
DE introduit en collecte ou héritage le corpus-père ou fils
PAR introduit en collecte ou héritage la variable numérique qui sert de clef externe

SOMMAIRE

Définitions 2
Qu'est-ce qu'une variable formulée 2

La page FORMULES 3

Le langage des formules 6
Mécanismes fondamentaux 6
Arithmétique élémentaire 8
Fonctions sur les nombres 9
Aléatoires et divers numériques 10
Opérations logiques élémentaires 10
Opérations sur les textes 11
Conversions 13
Fonctions auto-référentes 13
Comparaisons et prédicats 14
Séries 15
Sommations 16
Bases de données relationnelles 18
Structures conditionnelles 20
Qu'est-ce qu'une formule correcte ? 20
Formules multiples et sur collections 23

Le fonctionnement du langage : exemples commentés 26
Exemple de récursivité : 26
Recherche d'erreurs 27
Problèmes classiques et moins classiques 29

Inventaire récapitulatif des mots-clefs 32